Multi-fidelity Analysis for Human-in-the-loop Search and Tracking

Tomonari Furukawa

Final Report for a Project Funded by
Asian Office of Aerospace Research and Development

Computation Mechanics and Robotics Group
and
ARC Centre of Excellence for Autonomous Systems

School of Mechanical and Manufacturing Engineering
University of New South Wales
Australia

June, 2008

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
02 JUL 2008	**Final**	**19-09-2007 to 19-03-2008**

4. TITLE AND SUBTITLE
Multi-Fidelity Analysis for Human-in-the-Loop Search and Tracking

5a. CONTRACT NUMBER
FA48690714099

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Tomonari Furukawa

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
The University of New South Wales,Univ of New South Wales,Sydney NSW ,Australia,NA,2052

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AOARD, UNIT 45002, APO, AP, 96337-5002

10. SPONSOR/MONITOR'S ACRONYM(S)
AOARD

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AOARD-074099

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The aim of this project is to achieve multi-fidelity analysis for the multi-objective control of a team of unmanned aerial vehicles (UAVs) with humans in the loop of the estimation and control framework. The successful participation of humans in the loop for dynamically changing environments depends heavily on the reliability of data measured and information constructed by the UAVs. The multi-fidelity analysis proposed in the project enables real-time adjustment for computationally heavy search and tracking operations and further extracts quantities from the adjustment in the form of reliability.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**41**	

Multi-fidelity Analysis for Human-in-the-loop Search and Tracking

Tomonari Furukawa

Abstract

The aim of this project is to achieve multi-fidelity analysis for the multi-objective control of a team of unmanned aerial vehicles (UAVs) with humans in the loop of the estimation and control framework. The successful participation of humans in the loop for dynamically changing environments depends heavily on the reliability of data measured and information constructed by the UAVs. The multi-fidelity analysis proposed in the project enables real-time adjustment for computationally heavy search and tracking operations and further extracts quantities from the adjustment in the form of reliability.

Due to the termination of the project in six months for its renewal at Virginia Polytechnic Institute and State University, the aim has been scaled down to the real-time adjustment for search and tracking using a Graphics Processing Unit (GPU). The GPU, allowing grid-wise computation for image processing and analysis, is well suited to accelerate recursive Bayesian estimation which also requires computation in spatially discretized space for search and tracking. The GPU-based search and tracking system was initially implemented on a platform-in-the-loop simulator to enable real-time cooperative search and tracking in a virtual environment. The real-time capability of the system was then demonstrated with an unmanned ground vehicle (UGV) for its search operations.

Acknowledgements

This work is supported by the US Asian Office of Aerospace Research and Development (AOARD). We would like to express our sincere gratitude to Drs. William Nace and Ken Goretta at AOARD. The appreciation also goes to ARC Centre of Excellence for Autonomous Systems funded by the Australian Research Council (ARC) and the New South Wales State Government and its Director, Prof. Hugh F. Durrant-Whyte.

List of Papers

The papers produced during the half a year period are listed below. Due to the shortness of the period of the project, most of work produced remains unpublished. The additional list describes journal papers to be written with the progress made up to date, and approximately two conference papers will be published for the contents of each journal paper.

Papers (January - June, 2008)

Journal papers

1. Chern Ferng Chung and Tomonari Furukawa, "Coordinated Pursuer Control Using Particle Filters for Autonomous Search-and-Capture, Journal of Robotics and Autonomous Systems, in print.

2. Benjamin Lavis, Tomonari Furukawa and Hugh F. Durrant-Whyte, "Spatially Adaptive Exploration for Autonomous Bayesian Search and Tracking," Autonomous Robots, 24, pp. 387-399, 2008.

3. Lin Chi Mak and Tomonari Furukawa, "A Time-of-Arrival-Based Positioning Technique With Non-Line-of-Sight Mitigation Using Low-Frequency Sound", Advanced Robotics, Vol. 22, no. 5, pp. 507-526, 2008.

4. Lin Chi Mak, Mark Whitty and Tomonari Furukawa, "A localisation system for an indoor rotary-wing MAV using blade mounted LEDs", Sensor Review, Emerald, Vol. 28, Issue 2, pp. 125-131, 2008.

Conference papers

1. Benjamin Lavis and Tomonari Furukawa, "HyPE: Hybrid Particle-Element Approach for Recursive Bayesian Searching-and-Tracking," Robotic Systems and Science, Zurich, Switzerland, June 25-28, 2008, 8 pages, in print.

2. Benjamin Lavis, Yasuyoshi Yokokohji and Tomonari Furukawa, "Estimation and Control for Cooperative Autonomous Searching in Crowded Urban Emergencies," International Conference on Robotics and Automation, Pasadena, CA, May 19-23, 2008, pp. 2831-2836, 2008.

3. Lin Chi Mak, et al., "Design and development of the Micro Aerial Vehicles for Search, Tracking And Reconnaissance (MAVSTAR) for MAV08," 1st US-Asian demonstration and assessment of micro-aerial and unmanned ground vehicle technology (MAV08), Agra, India, Mar. 10-15, 2008.

4. Jamie Kelly, Makoto Kumon, Benjamin Lavis and Tomonari Furukawa, "Real-time Recursive Bayesian Estimation Using a Graphics Processing Unit and its Application to Micro Aerial Vehicles," 8th International Conference on Cooperative Control and Optimization (CCO08), Gainesville, FL, January 30-February 2, 2008, Abstract only.

Papers to be Written with Progress up to Date

1. Journal Paper 1: Real-time Adjustment for Belief-driven Search and Tracking Using Graphics Processing Unit.

2. Journal Paper 2: Multi-fidelity Analysis for Real-time Belief-driven Search and Tracking.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The control objectives of a group of sensor platforms can be classified into search and tracking depending on whether the target of concern can be detected or not. Whilst search is performed when the target is not found, the sensor platform which found a target tracks it to carry out desired tasks. Under the presence of uncertainties, it is expected to formulate the search and tracking in a probabilistic manner by utilizing a recursive Bayesian estimator.

Due to the difference in operation, recursive Bayesian estimators for search and tracking however used different numerical techniques in the past. Techniques effective for the search operation include the grid-based method, the element-based method and the Monte Carlo method. The strength of these methods for search clearly lies in their capability of representing the entire search space, but the consideration of the entire search space rendered the methods uninteresting to tracking where only the target PDF with a high, sharp peak, described only in a small region around the location of the found target, is concerned. The techniques suitable for tracking, such as the extended Kalman Filter (EKF), the sequential Monte Carlo (SMC) methods also known as the particle filter methods, the sequential Quasi-Monte Carlo (SQMC) methods and their variants, result from their computational efficiency in describing the PDF of the observable target. As a result, the areas considered in the target space are limited to those where targets are observable, and the past work treated search and tracking independently.

In this report, real-time adjustment using a Graphics Processing Unit (GPU)

has been presented for the recursive Bayesian search and tracking. The numerical technique adopted for search and tracking is the grid-based method, which shows its superiority in the representation of search space and inferiority in computation time, particularly for the tracking operation. By analyzing its grid-wise computational process, its implementation on a GPU allows real-time recursive Bayesian estimation, thereby removing the bottleneck of the grid-based method.

The report is organized as follows. The following chapter reviews the recursive Bayesian estimation and the grid-based method whilst Chapter 3 refers to the fundamentals of GPUs. Chapter 4 presents the proposed implementation of the recursive Bayesian estimation on the GPU, and experimental results demonstrating its effectiveness are dealt with in Chapter 5.

Chapter 2

Recursive Bayesian Estimation

This chapter describes the fundamentals of the recursive Bayesian estimation when a single autonomous vehicle is concerned with a single target.

2.1 Target and Sensor Platform Models

Consider a target t of concern, the motion of which is discretely given by

$$\mathbf{x}_{k+1}^t = \mathbf{f}^t\left(\mathbf{x}_k^t, \mathbf{u}_k^t, \mathbf{w}_k^t\right) \tag{2.1}$$

where $\mathbf{x}_k^t \in \mathcal{X}^t$ is the state of the target at time step k, $\mathbf{u}_k^t \in \mathcal{U}^t$ is the set of control inputs of the target, and $\mathbf{w}_k^t \in \mathcal{W}^t$ is the "system noise" of the target. In the aforementioned marine SAR scenario, the target state describes the two-dimensional position of the life raft, whilst the target control inputs and the system noise correspond to the wind and current acting in two-dimensional space and their uncertainties or disturbances, respectively.

In order for the formulation of the SAT problem, this moving target is searched and tracked by a vehicle s, the global state of which is assumed to be accurately known by the use of sensors such as GPS, a compass and an IMU. The motion model is thus given by

$$\mathbf{x}_{k+1}^s = \mathbf{f}^s\left(\mathbf{x}_k^s, \mathbf{u}_k^s\right) \tag{2.2}$$

where $\mathbf{x}_k^s \in \mathcal{X}^s$ and $\mathbf{u}_k^s \in \mathcal{U}^s$ represent the state and control input of the vehicle,

respectively. The vehicle also carries a sensor with an "observable region" as its physical limitation to observe a target. The observable region is determined not only by the properties of the sensor such as signal intensity but also the properties of the target such as the reflectivity. Defining the probability of detection $0 \leq P_d\left(\mathbf{x}_k^t|\mathbf{x}_k^s\right) \leq 1$ from these factors as a reliability measure for detecting the target t, the observable region can be expressed as

$$ {}^s\mathcal{X}_o^t = \left\{\mathbf{x}_k^t | 0 < P_d\left(\mathbf{x}_k^t|\mathbf{x}_k^s\right) \leq 1\right\} $$

Accordingly, the target state observed from the sensor platform, ${}^s\mathbf{z}_k^t \in \mathcal{X}^t$, is given by

$$ {}^s\mathbf{z}_k^t = \begin{cases} {}^s\mathbf{h}^t\left(\mathbf{x}_k^t, \mathbf{x}_k^s, {}^s\mathbf{v}_k^t\right) & \mathbf{x}_k^t \in {}^s\mathcal{X}_o^t \\ \emptyset & \mathbf{x}_k^t \notin {}^s\mathcal{X}_o^t \end{cases} \tag{2.3} $$

where ${}^s\mathbf{v}_k^t$ represents the observation noise, and $\emptyset$ represents an "empty element", indicating that the observation contained no information on the target or that the target is unobservable when it is not within the observable region. Note here that the terms "sensor platform" and 'vehicle' are used interchangeably in this report as the vehicle is assumed to carry only one sensor for observation.

2.2 Recursive Bayesian Estimation

Recursive Bayesian estimation forms a basis to the estimation of nonlinear non-Gaussian stochastic models. Let a sequence of states of the sensor platform s and a sequence of observations by the sensor platform from time step 1 to time step k be $\tilde{\mathbf{x}}_{1:k}^s \equiv \{\tilde{\mathbf{x}}_i^s | \forall i \in \{1, ..., k\}\}$ and ${}^s\tilde{\mathbf{z}}_{1:k} \equiv \{{}^s\tilde{\mathbf{z}}_i | \forall i \in \{1, ..., k\}\}$, respectively. Note here that $\tilde{(\cdot)}$ represents an instance of variable $(\cdot)$. Given a prior density of the target $p\left(\tilde{\mathbf{x}}_0^t\right)$ and sequences of states $\tilde{\mathbf{x}}_{1:k}^s$ and observations ${}^s\tilde{\mathbf{z}}_{1:k}$, the PDF of the target at any time step k, $p\left(\mathbf{x}_k^t|{}^s\tilde{\mathbf{z}}_{1:k}, \tilde{\mathbf{x}}_{1:k}^s\right)$, can be estimated recursively through the two stage equations, update and prediction:

1. **Update**: The update equation computes the posterior density $p\left(\mathbf{x}_k^t|{}^s\tilde{\mathbf{z}}_{1:k}, \tilde{\mathbf{x}}_{1:k}^s\right)$ given the corresponding state estimated with the observations up to the pre-

vious time step $p\left(\mathbf{x}_k^t|{}^s\tilde{\mathbf{z}}_{1:k-1}, \tilde{\mathbf{x}}_{1:k}^s\right)$ and a new observation ${}^s\tilde{\mathbf{z}}_k$. The equation is derived by applying formulae for marginal distribution and conditional independence and given by

$$p\left(\mathbf{x}_k^t|{}^s\tilde{\mathbf{z}}_{1:k}, \tilde{\mathbf{x}}_{1:k}^s\right) = \frac{l\left(\mathbf{x}_k^t|{}^s\tilde{\mathbf{z}}_k, \tilde{\mathbf{x}}_k^s\right) p\left(\mathbf{x}_k^t|{}^s\tilde{\mathbf{z}}_{1:k-1}, \tilde{\mathbf{x}}_{1:k}^s\right)}{\int_{\mathcal{X}^t} l\left(\mathbf{x}_k^t|{}^s\tilde{\mathbf{z}}_k, \tilde{\mathbf{x}}_k^s\right) p\left(\mathbf{x}_k^t|{}^s\tilde{\mathbf{z}}_{1:k-1}, \tilde{\mathbf{x}}_{1:k}^s\right) d\mathbf{x}_{k-1}^t}, \tag{2.4}$$

where $l\left(\mathbf{x}_k^t|{}^s\tilde{\mathbf{z}}_k, \tilde{\mathbf{x}}_k^s\right)$ represents the likelihood of $\mathbf{x}_k^t$ given ${}^s\tilde{\mathbf{z}}_k$ and $\tilde{\mathbf{x}}_k^s$. Notice that the update at $k = 1$ is carried out by letting $p\left(\mathbf{x}_k^t|{}^s\tilde{\mathbf{z}}_{1:k-1}, \tilde{\mathbf{x}}_{1:k}^s\right) = p\left(\tilde{\mathbf{x}}_0^t\right)$.

2. **Prediction**: The prediction step computes the PDF of the next state $p\left(\mathbf{x}_{k+1}^t|{}^s\tilde{\mathbf{z}}_{1:k}, \tilde{\mathbf{x}}_{1:k}^s\right)$ from the PDF in the current time step $p\left(\mathbf{x}_k^t|{}^s\tilde{\mathbf{z}}_{1:k}, \tilde{\mathbf{x}}_{1:k}^s\right)$. The prediction is carried out by Chapman-Kolmogorov equation, which is better known as Total Probability Theorem:

$$p\left(\mathbf{x}_{k+1}^t|{}^s\tilde{\mathbf{z}}_{1:k}, \tilde{\mathbf{x}}_{1:k}^s\right) = \int_{\mathcal{X}^t} p\left(\mathbf{x}_{k+1}^t|\mathbf{x}_k^t\right) p\left(\mathbf{x}_k^t|{}^s\tilde{\mathbf{z}}_{1:k}, \tilde{\mathbf{x}}_{1:k}^s\right) d\mathbf{x}_k^t, \tag{2.5}$$

where $p\left(\mathbf{x}_{k+1}^t|\mathbf{x}_k^t\right)$ is a probabilistic Markov motion model defined by Eq. (2.1) which maps the probability of transition from the current state $\mathbf{x}_k^t$ to the next state $\mathbf{x}_{k+1}^t$.

In their numerical implementation, the update and prediction processes, as described in Equations (2.4) and (2.5), essentially require the evaluation of a function at an arbitrary point in the target space $\mathcal{X}^t$, $f(\mathbf{x}^t)$, and the integration of a function over the target space, $I = \int_{\mathcal{X}^t} f(\mathbf{x}^t) d\mathbf{x}^t$. As described in Sec. 1, the search techniques achieve this by spreading nodes for integration over the target space whereas the tracking techniques only consider the subspace where a non-zero value of the target PDF appears.

2.3 Grid-based Method

Since the majority of the numerical techniques for recursive Bayesian estimation compute on the nodal discrete basis, the superiority of the proposed element-based method in continuous representation to the techniques cannot be easily discussed in

a quantitative manner. As the grid-based method can represent the target space continuously by considering grid cells rather than grid points, this chapter will present the continuous formulation of the grid-based method, which describes the target PDF in terms of the step function rather than the Dirac delta functions [**?**], and address the limitations of the grid-based method.

2.3.1 Representation of Target Space

Figure 2.1 illustrates the process that numerically derives the approximate target space using the grid-based method when a two-dimensional circular target space is concerned. The numerical approximation involves the following three steps:

1. Creation of a rectangular space which covers the entire target space,
2. Representation of the rectangular space as a grid,
3. Selection of grid cells that represent the target space.

In order to avoid the complication of formulations, let the target space be represented two-dimensionally as $\mathbf{x}^t = [x^t, y^t]^\top \in \mathcal{X}^t$. The creation of a rectangular space $\mathcal{X}^r$ that covers the target space is achieved by firstly defining the minimum and maximum values of the target space

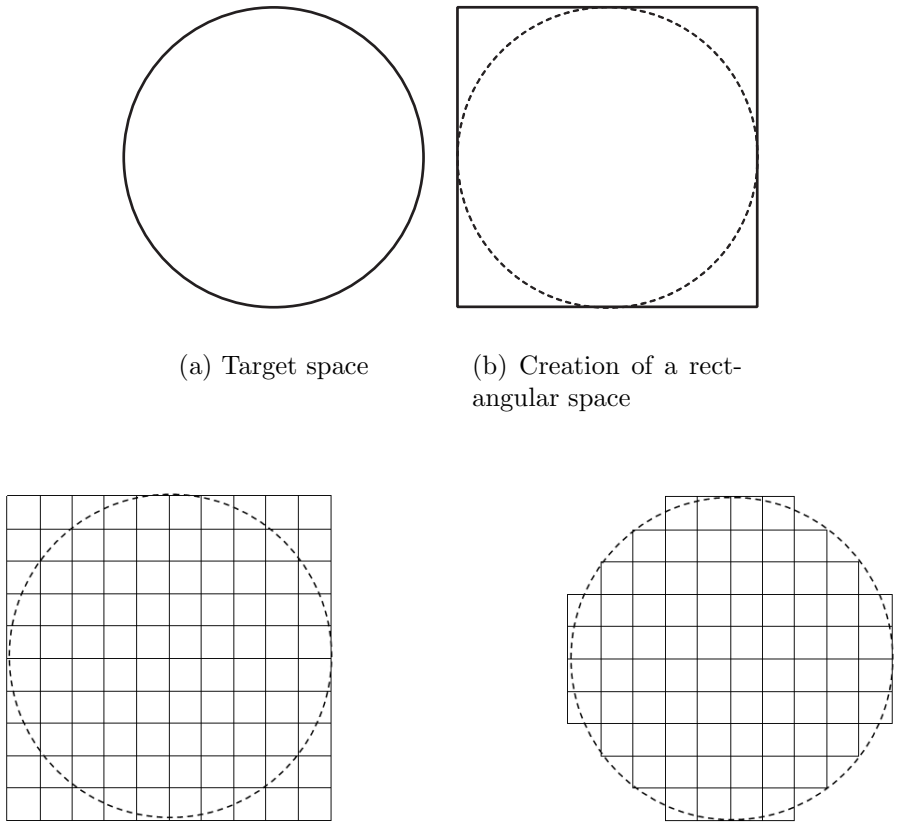
$$x^t_{\min} = \min\left\{x^t\right\}, x^t_{\max} = \max\left\{x^t\right\}$$
$$y^t_{\min} = \min\left\{y^t\right\}, y^t_{\max} = \max\left\{y^t\right\}$$

and then creating a rectangular space as

$$\mathcal{X}^r \equiv \left\{\mathbf{x} | \forall x \in [x^t_{\min}, x^t_{\max}), \forall y \in [y^t_{\min}, y^t_{\max})\right\} \supseteq \mathcal{X}^t$$

where $\mathbf{x} = [x, y]^\top$. The grid space is further introduced by discretizing the rectangular space by n_x and n_y. In order to do so, the dimensions of a grid cell are defined as

$$\Delta x^r = \frac{(x^t_{\max} - x^t_{\min})}{n_x}, \Delta y^r = \frac{(y^t_{\max} - y^t_{\min})}{n_y}. \tag{2.6}$$



(a) Target space

(b) Creation of a rectangular space

(c) Creation of a grid

(d) Selection of grid cells

Fig. 2.1: Grid-based method

This results in introducing the center of each grid cell as

$$\bar{\mathbf{x}}_{i,j}^{r} = \left[(i-0.5)\Delta x^{r} + x_{\min}^{t}, (j-0.5)\Delta y^{r} + y_{\min}^{t}\right]^{\top}, \quad \forall i \in \{1,...,n_x\}, \forall j \in \{1,...,n_y\}, \tag{2.7}$$

and defining each grid cell as

$$\mathcal{X}_{i,j}^{r} = \left\{ \mathbf{x} \middle| \, |x - \bar{x}_i^r| < \frac{1}{2}\Delta x^r, \left|y - \bar{y}_j^r\right| < \frac{1}{2}\Delta y^r \right\}.$$

Note that $\cup_{i=1}^{n_{xx}} \cup_{j=1}^{n_{yy}} \mathcal{X}_{i,j}^{r} = \mathcal{X}^{r}$ and $\cap_{i=1}^{n_{xx}} \cap_{j=1}^{n_{yy}} \mathcal{X}_{i,j}^{r} = \emptyset$. The selection of grid cells that represent the target space can be performed in various ways. One easy way is to select a grid cell if its center is located in the target space:

$$\mathcal{X}_{i,j}^{r} \subset \mathcal{X}^{t} \quad \text{if} \quad \bar{\mathbf{x}}_{i,j}^{r} \in \mathcal{X}^{t} \ .$$

The resulting configuration of the target space in the figure, with the rectangular space partitioned by $n_x = n_y = 10$, depicts the coarse representation of the boundary of the target space, which is one of the deficiencies of the grid-based method.

2.3.2 Evaluation of Function and its Integral

Let the approximate target space derived by the processes described in the last section be

$$\mathcal{X}^{t} \approx \mathcal{X}^{g} \equiv \left\{ \mathcal{X}_1^g, ..., \mathcal{X}_{n_g}^g \right\}$$

with the center of each grid cell being $\bar{\mathbf{x}}_i^g$, $\forall i \in \{1,...,n_g\}$, where n_g is the number of grid cells approximating the target space. Having the approximate function defined over the target space be $f^g : \mathcal{X}^g \to R$, the value of the function given a point in the target space, $\mathbf{x}^t = \tilde{\mathbf{x}}^t \in \mathcal{X}^g$, can be approximately computed as

$$f(\tilde{\mathbf{x}}^t) \approx f^g(\tilde{\mathbf{x}}^t) \equiv \sum_{i=1}^{n_g} f(\bar{\mathbf{x}}_i^g)\delta_i\left(\tilde{\mathbf{x}}^t - \bar{\mathbf{x}}_i^g\right), \tag{2.8}$$

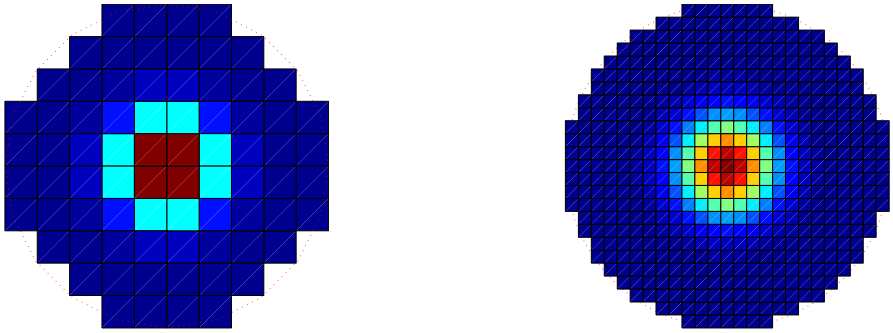
where the indicator function $\delta_i(\cdot)$ is defined as

$$\delta_i\left(\mathbf{x}^t - \bar{\mathbf{x}}_i^g\right) = \begin{cases} 1 & \mathbf{x}^t \in \mathcal{X}_i^g \\ 0 & \text{Otherwise} \end{cases} . \tag{2.9}$$

Using the representation of the approximate function, the integral of the function is given by

$$\begin{aligned} I &= \int_{\mathcal{X}^t} f(\mathbf{x}^t) d\mathbf{x}^t \\ &\approx \int_{\mathcal{X}^g} f^g(\mathbf{x}^t) d\mathbf{x}^t = \Delta x^r \Delta y^r \sum_{i=1}^{n_g} f(\bar{\mathbf{x}}_i^g). \end{aligned} \tag{2.10}$$

Figures 2.2(a) and (b) show Gaussian distributions when the circular target space is partitioned by 10 and 20, respectively. In addition to the target space, the distribution is also represented coarsely if the number of partitions is small.



(a) 10 partitions (b) 20 partitions

Fig. 2.2: Function with grid-based method

Chapter 3

Graphics Processor Unit

The GPU is a powerful tool for programmers with a great deal of potential for solving the problems faced by scientists and engineers who are attempting to achieve Real-Time performance in their systems. The chapter first presents the advantages of using a workhorse GPU for parallel computation by examining recent trends in computer hardware. The chapter then describes the architecture and operation of NVidia GPUs and how they can be used for general purpose computing by referring to their use with recursive Bayesian estimation.

3.1 Real-Time Computational Limits

The improvement in computational power of GPUs in the past 5 years has far outstripped that of CPU based systems. The operational demands placed on GPUs by the gaming industry have dictated a different architecture to CPUs based on the use of multiple processors to perform simultaneous calculations of multiple results in a single operation. It is important to understand these differences so that parallel algorithms can be developed which are suitable for implementation using it.

A typical CPU based system is shown in Figure 3.1. It consists of a controller, Arithmatic Logic Units (ALUs) to compute the most basic operations on input data, cache memory which stores immediate inputs and outputs from the ALUs, and Dynamically Randomly Allocated Memory (DRAM) which holds the input and output data for the CPU operation. A typical CPU operation is as follows:

1. CPU loads entire input data variable into DRAM;
2. CPU loads small subset of input data into cache which is required for first part of operation;
3. ALU performs required algorithm on input data in cache and places resulting data back in cache;
4. CPU outputs result from cache to DRAM;
5. CPU loads new piece of data from DRAM to cache and repeats until entire operation is complete; and
6. CPU outputs entire output data variable from DRAM.

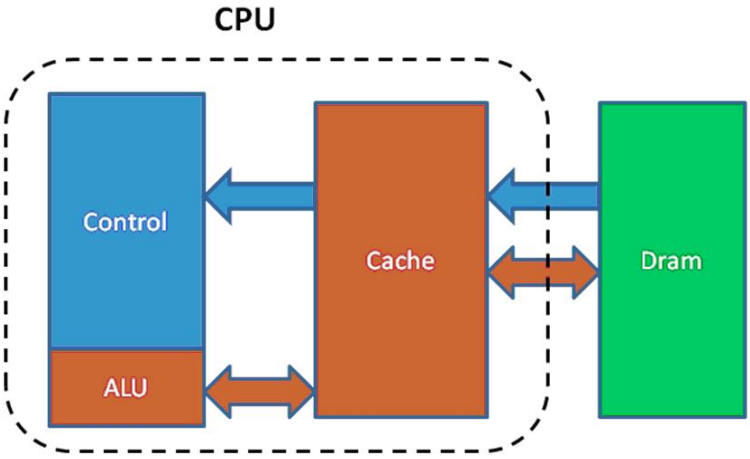


Fig. 3.1: Central Processing Unit Architecture

This linear structure basically iterates the required operation over the entire set of input data one piece at a time which severely limits the efficiency of the computation. In a GPU based operation, instead of having a single processor operation, there are multiple ALUs each with their own control and cache as shown in Figure 3.1 , which performs the same operation as above. The operation is now performed on batches of data as opposed to one at a time and can lead to dramatic reductions in the time taken to complete each operation.

The GPU architecture results in a speeding up of the arithmetic part of the operation, at the cost of increase in memory operations as more data and control

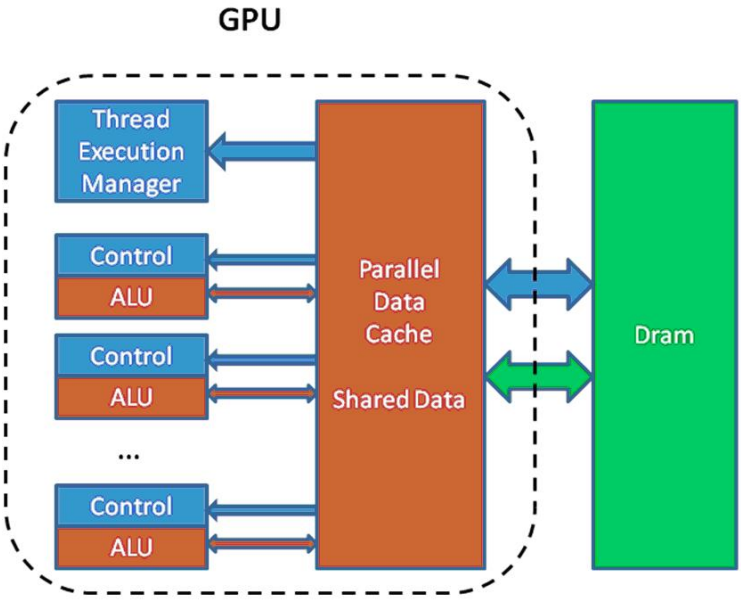


Fig. 3.2: Graphics Processing Unit Architecture

information needs to be sent every iteration. Thus operations which lend themselves to such architecture have a high arithmetic intensity, or ratio of arithmetic operations to memory operations. The most efficient algorithms for this are the Single Input Multiple Data (SIMD) operations, as only a single control needs to be sent to all of the controllers for the ALUs and each ALU simply performs the operation on the piece of data input to it.

3.2 Compute Unified Device Architecture

The most recent GPUs produced by NVidia are designed to operate with a C based library called Compute Unified Device Architecture (CUDA). This very basically consists of C based functions which allow a CPU to control general computational operations on the GPU. Such operations which must be controlled included:

- The creation of variable and memory locations on the GPU to store input and output data;
- The transfer of input data to the GPU;
- The general processing of input data by the GPU as required by the algorithms

being applied; and

- The transfer of output data back from the GPU.

These functions can be included in any C based code so that standard programs can utilize the GPU as necessary to speed up computation.

NVidia GPUs, such as the G80 chipset GPUs, when programmed using the CUDA functions are in essence highly multithreaded coprocessors. That is, the GPU is a computing device which may be controlled by functions on the CPU to perform operations on data in parallel. The program which performs a portion of the algorithm a number of times on different pieces of data is known as a kernel and is downloaded to the GPU prior to or during run-time.

Figure 3.2 illustrates the execution of the kernel on the GPU using batches of "threads", each of which will compute a single operation on one part of the input data. The threads are characterized and processed as follows:

- Threads are grouped together into blocks, in which each thread cooperates to perform the required computation, sharing memory as necessary to ensure efficient and rapid operation;

- In each thread block, the threads are identified by its *thread ID* which refers to its position inside the block;

- The blocks may be created in a two-dimensional fashion with a width and height and total number of threads;

- The number of threads each block can contain is limited by the memory architecture of the individual GPU. Batching groups of blocks of the same size together can be used to create larger size kernels for performing larger operations;

- Synchronization of the threads operation can only be performed within blocks, which is necessary to ensure that all of the operations requested of the threads have been performed before the results are output from the GPU

- Blocks have no ability to share memory or synchronize information and may be run either sequentially or in parallel by the GPU; and
- Blocks have their own identification number in the same way that threads do.

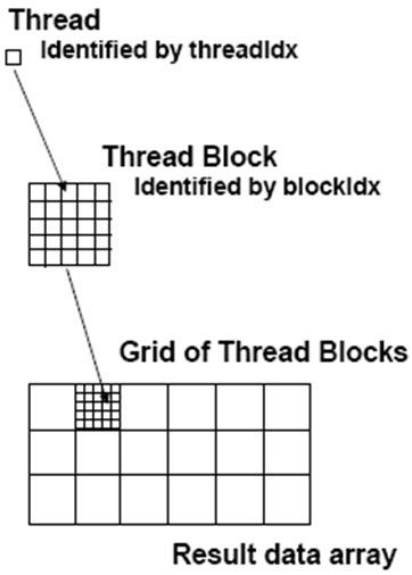


Fig. 3.3: Thread and Grid Block Layout for NVidia GPU

Figure 3.2 shows the memory architecture of the G80 chipset NVidia GPUs which consists of the following memory locations:

- Global, constant and texture memory spaces which are part of each grid for the kernel and are accessible by the threads (global memory can be read and written two, constant and texture memory can only be written);
- Local memory internal to each block;
- Registers for each thread to use; and
- Shared memory which all threads within each block can read and write to.

The multiprocessors on the GPU are able to schedule and execute a batch of blocks, in a series. If the shared memory allocated to each block can reside in the on-chip shared memory, the memory access speed is improved dramatically. Active or executing batches of blocks in a multi-processor are divided into groups of threads

called Warps of the same number of threads. Warps of threads are periodically switched by a thread scheduler to maximize the processors computational efficiency.

As has been stated above, the CUDA programming library allows programmers to complete access to the operation of the NVidia GPU, which includes allocation of variables in the memory locations shown above. The library itself consists of two parts:

1. Extensions to the C language itself to allow the programmer to target specific portions of the code to execution on the GPU rather than the CPU; and
2. Three sections of run-time library:
    - Host component to perform normal operation on the CPU;
    - Device component to run required operations on the GPU; and
    - Common component to allow for data transfer and control by the CPU of the GPU.

This allows a general user to frame their algorithms in such a manner that they may be computed as a separate function on the GPU and passed back for use by the C program on the CPU, which is how it is to be used for recursive Bayesian estimation. A proper understanding of this architecture is essential for using the NVidia GPU effectively and efficiently so that gains in computation time may be made.

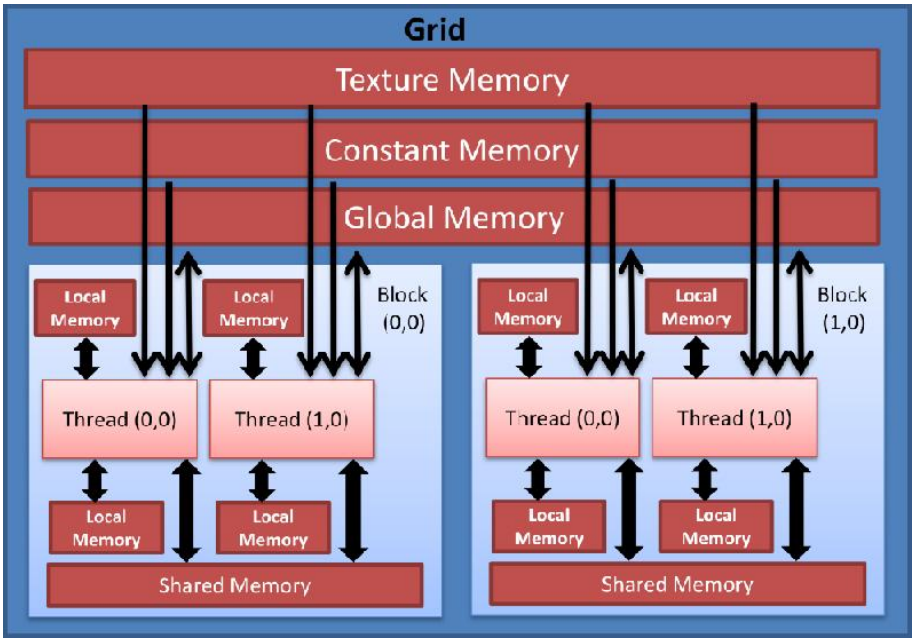


Fig. 3.4: Memory Architecture of an NVidia GPU

Chapter 4

Implementation of Recursive Bayesian Estimation on GPU

Having the recursive processes carried out on a grid basis, the recursive Bayesian estimation is well suited to the processing on the GPU. This chapter presents the implementation of recursive Bayesian estimation on a GPU and its estimation for maximum improvement in performance. Effort was made to understand how input parameters such as the search space size effect the computational improvement so that this conclusion is valid for typical search space sizes.

4.1 Data Parallelization

The approach chosen to improve the computational efficiency of the recursive Bayesian estimation is the form of parallelization known as data parallelization. Data parallelization is the process of converting processes which operate on each piece of data individually in a sequence, to one that operates on groups of data at once, which can result in a dramatic reduction in computation time. The recursive Bayesian estimation, described in Chapter 2, has multiple processes which lend themselves to computational speed-up by parallelization. However, their numerical implementation should be analyzed to determine which of these are likely to produce meaningful improvements to the computational speed.

The potential computational speedup produced by parallelizing operations in

a process can be estimated using Amdahl's law, which states that the maximum speedup achievable by parallelization is:

$$S = \frac{t_p}{t_o} = \frac{1}{(1-P) + \frac{P}{N}} \tag{4.1}$$

where S is the computational speedup, t_p is the iteration time for the parallelized algorithm, t_o is the iteration time for the original algorithm, P is the proportion of the process which is parallelizable and N is the number of parallel processors available to perform the parallelization. This formula is the theoretical maximum of the computational speedup and does not take into account the bottlenecking effect of communications between large numbers of threads; however it gives a good estimate of the effectiveness of parallelizing each operation in the recursive Bayesian estimation.

To enable this estimation, each operation in the recursive Bayesian estimation described in Chapter 2 must be examined to estimate the number of floating point operations. These results can be used to estimate the proportion of the algorithm that can be parallelized and allow the maximum potential speedup to be calculated using Equation (4.1).

4.2 Parallelization of Recursive Bayesian Estimation

As was described in Chapter 2, the grid-based representation of the target space involved describing the target space as a set of grid cells. The probability density function for the grid-based representation is a matrix of probability values each of which represents the probability of locating the target inside the corresponding grid cell. There is a significant advantage in representing the target space in this manner when formulating parallel algorithms for recursive Bayesian estimation, as the organization of the data itself is directly related to the physical location of the grid cells in the target space. This means very low overhead due to memory operations to determine the physical relationships between grid cells when performing the pre-

diction step, which is one of the major factors in the effectiveness of parallelization. If threads operating on different pieces of data have to perform multiple memory operations to determine which piece of data to operate on, the computational improvements due to parallelization are significantly reduced. Both the update and the prediction processes have been examined in this section to determine the effectiveness of implementing them in parallel on the NVidia GPU.

4.2.1 Update

The update operation requires the computation of Equation 2.4 in a discretized space with grid cells. Let the probability density function of the grid cell $[i, j]$ at the kth iteration be $p^{i,j}_{\mathbf{x}^t_k}(\cdot)$. Given the prior probability density function $p^{i,j}_{\mathbf{x}^t_k}({}^s\tilde{\mathbf{z}}_{1:k-1}, \tilde{\mathbf{x}}^s_{1:k})$ and the observation likelihood $l^{i,j}_{\mathbf{x}^t_k}({}^s\tilde{\mathbf{z}}_k, \tilde{\mathbf{x}}^s_k)$, the probability density function of the grid cell $[i, j]$ can be updated as

$$p^{i,j}_{\mathbf{x}^t_k}({}^s\tilde{\mathbf{z}}_{1:k}, \tilde{\mathbf{x}}^s_{1:k}) = \frac{q^{i,j}_{\mathbf{x}^t_k}({}^s\tilde{\mathbf{z}}_{1:k}, \tilde{\mathbf{x}}^s_{1:k})}{\Delta x_r \Delta y_r \sum_{\alpha=1}^{n_x}\sum_{\beta=1}^{n_y} q^{\alpha,\beta}_{\mathbf{x}^t_k}({}^s\tilde{\mathbf{z}}_{1:k}, \tilde{\mathbf{x}}^s_{1:k})}, \quad \forall\alpha \in \{1, ..., n_x\}, \forall\beta \in \{1, ..., ny\}, \tag{4.2}$$

where

$$q^{i,j}_{\mathbf{x}^t_k}({}^s\tilde{\mathbf{z}}_{1:k}, \tilde{\mathbf{x}}^s_{1:k}) = l^{i,j}_{\mathbf{x}^t_k}({}^s\tilde{\mathbf{z}}_k, \tilde{\mathbf{x}}^s_k)\, p^{i,j}_{\mathbf{x}^t_k}({}^s\tilde{\mathbf{z}}_{1:k-1}, \tilde{\mathbf{x}}^s_{1:k}), \forall i \in \{1, ..., n_x\}, \forall j \in \{1, ..., n_y\} \tag{4.3}$$

In the numerical implementation, the update operation can be broken down into three steps:

1. Calculate $q^{i,j}_{\mathbf{x}^t_k}({}^s\tilde{\mathbf{z}}_{1:k}, \tilde{\mathbf{x}}^s_{1:k})$ by multiplying the predicted probability density function $p^{i,j}_{\mathbf{x}^t_k}({}^s\tilde{\mathbf{z}}_{1:k-1}, \tilde{\mathbf{x}}^s_{1:k})$ by the observation likelihood $l^{i,j}_{\mathbf{x}^t_k}({}^s\tilde{\mathbf{z}}_k, \tilde{\mathbf{x}}^s_k)$;
2. Sum $\sum_{\alpha=1}^{n_x}\sum_{\beta=1}^{n_y} q^{\alpha,\beta}_{\mathbf{x}^t_k}({}^s\tilde{\mathbf{z}}_{1:k}, \tilde{\mathbf{x}}^s_{1:k})$; and
3. Calculate $p^{i,j}_{\mathbf{x}^t_k}({}^s\tilde{\mathbf{z}}_{1:k}, \tilde{\mathbf{x}}^s_{1:k})$ by dividing $q^{i,j}_{\mathbf{x}^t_k}({}^s\tilde{\mathbf{z}}_{1:k}, \tilde{\mathbf{x}}^s_{1:k})$ by $\sum_{\alpha=1}^{n_x}\sum_{\beta=1}^{n_y} q^{\alpha,\beta}_{\mathbf{x}^t_k}({}^s\tilde{\mathbf{z}}_{1:k}, \tilde{\mathbf{x}}^s_{1:k})$.

Out of the three steps, Steps 1 and 3 are the operations whose calculations can be conducted independently for every grid cell. Step 1 requires a multiplication

operation for each grid cell in the target space, thus for the search space with $n_x n_y$ cells or less, there are $n_x n_y$ floating point operations per iteration at most. This number could be much lower as there may be large parts of the search space where either $p^{i,j}_{\mathbf{x}^t_k}\left({}^s\tilde{\mathbf{z}}_{1:k-1}, \tilde{\mathbf{x}}^s_{1:k}\right)$ or $l^{i,j}_{\mathbf{x}^t_k}\left({}^s\tilde{\mathbf{z}}_k, \tilde{\mathbf{x}}^s_k\right)$ is zero. However for these calculations, the worst case scenario will be used as this is the most conservative estimate of the potential speedup. Step 2 requires additional floating operations through division, which totals $n_x n_y$ at most. Thus, the maximum number of floating point operations is estimated as $2n_x n_y$.

4.2.2 Prediction

The prediction operation requires the numerical evaluation of the Chapman-Kolmogorov equation described in Equation (2.5). Given the updated PDF $p^{ij}_{\mathbf{x}^t_k}\left({}^s\tilde{\mathbf{z}}_{1:k-1}, \tilde{\mathbf{x}}^s_{1:k}\right)$ as well as the Malkov motion model $p^{ij}_{\mathbf{x}^t_{k+1}|\mathbf{x}^t_k}$ constructed in the matrix form as the convolution kernel, the PDF can be predicted as

$$\begin{aligned} p^{i,j}_{\mathbf{x}^{t+1}_k}\left({}^s\tilde{\mathbf{z}}_{1:k}, \tilde{\mathbf{x}}^s_{1:k}\right) &= p^{i,j}_{\mathbf{x}^t_k}\left({}^s\tilde{\mathbf{z}}_{1:k}, \tilde{\mathbf{x}}^s_{1:k}\right) \otimes p^{i,j}_{\mathbf{x}^t_{k+1}|\mathbf{x}^t_k}, \\ &= \sum_{\alpha=0}^{I}\sum_{\beta=0}^{J} p^{\alpha,\beta}_{\mathbf{x}^t_{k+1}|\mathbf{x}^t_k} p^{i-\alpha,j-\beta}_{\mathbf{x}^t_k}\left({}^s\tilde{\mathbf{z}}_{1:k}, \tilde{\mathbf{x}}^s_{1:k}\right), \\ &\quad \forall i \in \{1, ..., n_x\}, \forall j \in \{1, ..., n_y\}, \end{aligned} \tag{4.4}$$

where $\otimes$ indicates the convolution of the updated PDF with the Malkov motion model. The equation first shows that the prediction operation at each grid cell can be conducted independently similarly to the update. However, it is also shown that the computation time for prediction is largely dominated by the size of the convolution kernel. Reducing the size will contribute to the real-time operation. However, the convolution kernel must be able to capture the motion of the target, so there will be a limitation in operational speed unless accuracy is sacrificed.

Figure 4.2.2 shows how the number of prediction steps varies with kernel radii and grid size where the kernel radius is defined as the number of grid cells the kernel matrix extends from the center cell to the outer row or column, not including the center cell itself. For a suitable kernel radius r_k, the number of multiplications

required to calculate each grid cell is $(2r_k + 1)^2$. For each grid cell, these multiplications are summed together, and thus there are a further $(2r_k + 1)^2$ floating point operations per cell. In total, the number of floating point operations in the Prediction step is $n_g(2(2r_k + 1)^2)$.

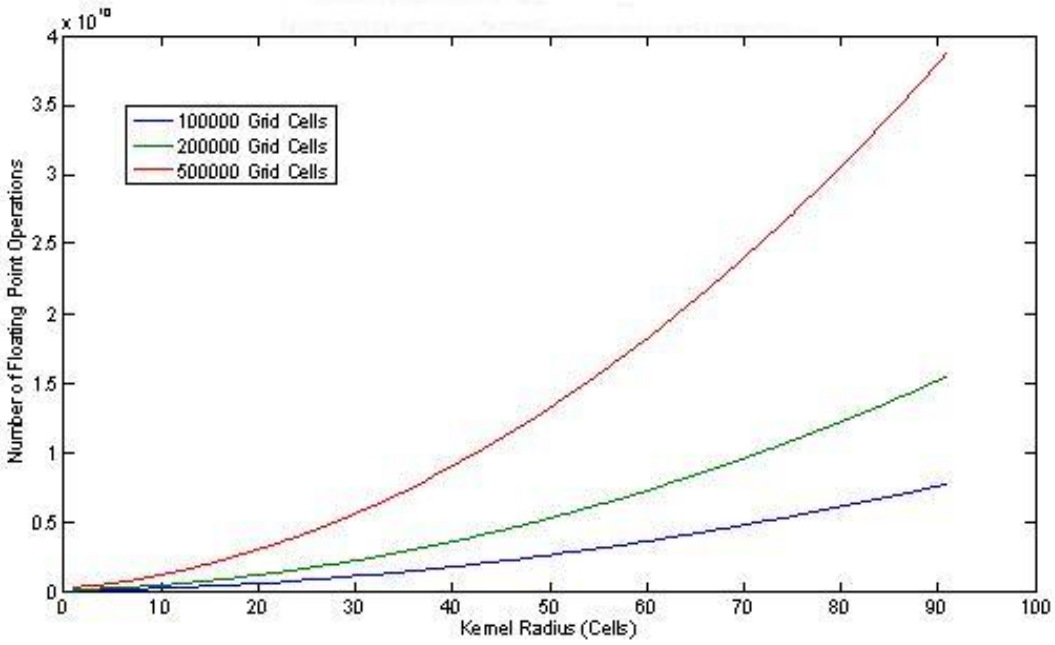


Fig. 4.1: Number of floating point operations in prediction

The figure shows clearly the exponential increase in operations with kernel radii. There are also other factors, which will determine the kernel size in the grid-based representation, including the allowable memory size and the transfer rates between different processors and memory locations.

4.2.3 Estimated Computational Speedup

Table 4.1 summarizes the approximate number of floating point operations in each major operation of the recursive Bayesian estimation. For a given number of grid cells, kernel radius and number of parallel processors available on the GPU the maximum attainable speedup may now be estimated with Equation 4.1 which has been plotted in Figure 4.2.3. Note that the number of grid cells is a factor of each of the total number of floating point operations in these processes, thus the speedup which calculates the proportion of time spent in each process, will be theoretically

independent of grid size. For example, for a grid with 1,000,000 cells, a kernel radius of 8 and using an NVidia G80 chipset GPU with 32 coprocessors, the maximum speedup attainable is approximately 27.58 for the prediction and 1.00 for the update step.

Table 4.1: Computational requirement for grid-based method

Operation	Number of floating point operations
Update	$3n_g$
Prediction	$n_g \left\{2\left(2r_k+1\right)^2\right\}$

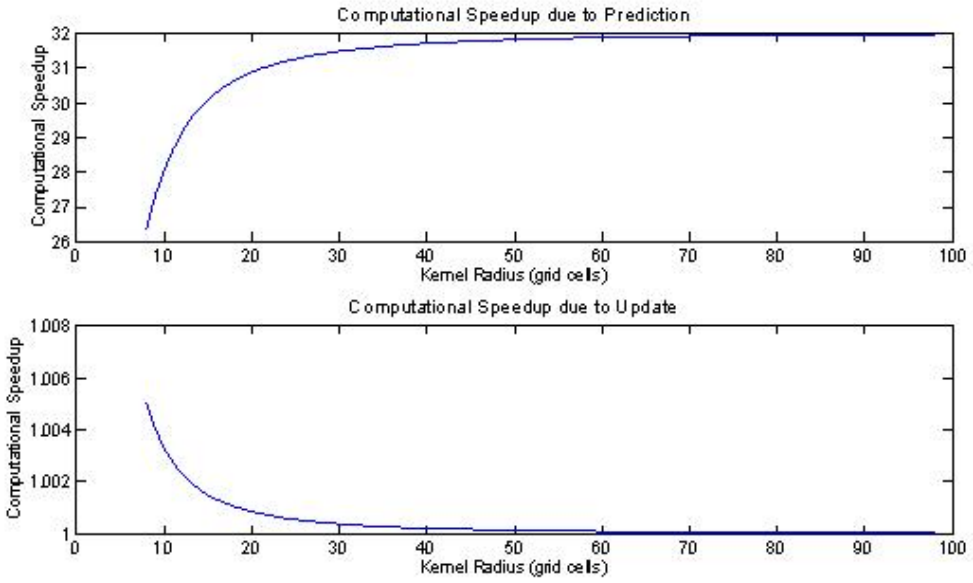


Fig. 4.2: Computational Speedup for Grid-based Representation

When computed linearly using a CPU, the computational time required for an iteration of recursive Bayesian estimation at 4740 grid cells was 1.27 seconds, which would mean that, by parallelizing the prediction step alone, the iteration time could be reduced to approximately 0.046 seconds with a 96.4% reduction in computation time. Although this is a theoretical limit, the parallelization of the prediction step with the grid-based method has very low overhead due to memory access issues. This is because the data is stored coherently in a matrix in the grid-based method, which means that the actual improvement in computation time should be close to this estimated limit.

Chapter 5

Experimental Results

This chapter presents results of a series of qualitative and quantitative tests carried out to investigate real-time capability of the system implementing the proposed recursive Bayesian estimation. The tests described below have been performed using problem parameters specified in the tables for each section. For the validation of the system operation and testing of its computational efficiency, simple internal inputs and outputs are used. The computational speeds measured for the GPU based parallel form of the recursive Bayesian estimation have been compared with that of the linear form to measure the speedup due to parallelization of the prediction step. This result is to be compared with the estimations made in Chapter 4 of the maximum improvement in computational time. Following these tests are two tests which demonstrate the system operation with external inputs and outputs, which include both simulations and real hardware. These tests demonstrate the efficacy of the developed system in real-time practical environments.

5.1 Experimental Setup and Methodology

The experimental setup used in the following tests is a recursive Bayesian estimation system in a Matlab environment which utilizes an NVidia GPU to achieve real-time performance. The speeds of GPUs have been increasing at almost four times the rate of that of CPUs. In order to demonstrate the computational speed improvements achieved using parallel algorithms on a GPU, it must be compared against a linear

algorithm running on a CPU.

The setup specifications which have been chosen is shown in Table 5.1. To prevent biasing of the results to either the CPU or GPU based implementation, the chosen CPU and GPU needs to be of equivalent standard in memory and speed, and both need to be current up to date versions with correctly installed drivers and clean installs of the required programs.

Table 5.1: Test computer system specifications

Processor	Intel Core2Duo, 2.4GHz
Memory	2.0GB RAM
Operating system	Windows XP Pro, Service Pack 2
GPU	NVidia 8800GTS, 640MB onboard RAM, stream processors
Matlab	R2007b with nvmex installed

The CPU and GPU chosen here were both top of the range units at the time of purchase. For the purposes of this project, these were chosen as benchmarks and are representative of current trends in CPU and GPU technology. The NVidia 8800 GTS supports CUDA version 1.1 which has the required Matlab compiler installed so that MEX files in Matlab may control the GPU. The GPU features a core clock speed of 500MHz, 640MB of onboard Ram and a significant memory bandwidth of 64GB/s. It has the CUDA functionality installed for use with C based MEX files through the Matlab 2007b interface.

5.2 Validation

5.2.1 Test 1: Validation of parallelized recursive Bayesian estimation with linear form

Test 1 was aimed at validating the parallelized form of recursive Bayesian estimation by comparing it with results from the linear form which were also computed using Matlab. Before the performance of the real-time recursive Bayesian estimation system can be studied, the underlying parallelized form of the recursive Bayesian estimation needs to be validated to ensure fidelity with the original form of the algorithm.

The target grid size must also be input to define the problem completely which is to be varied to examine how this affects the coherence between the linear and parallel forms of recursive Bayesian estimation. Each of these forms was iterated for a range of different grid sizes. As this process is iterative, any errors in an iteration of the parallel algorithm will cause a rapid drift between the probability density functions of the two different methods. In order to make meaningful comparisons between them the probability density function each method uses at the start of each iteration must be the same for both forms. The linear algorithm will always be taken to be the correct one at the beginning of each time step and both algorithms will be applied to it. This process is illustrated in the flow chart in Figure 5.2.1.

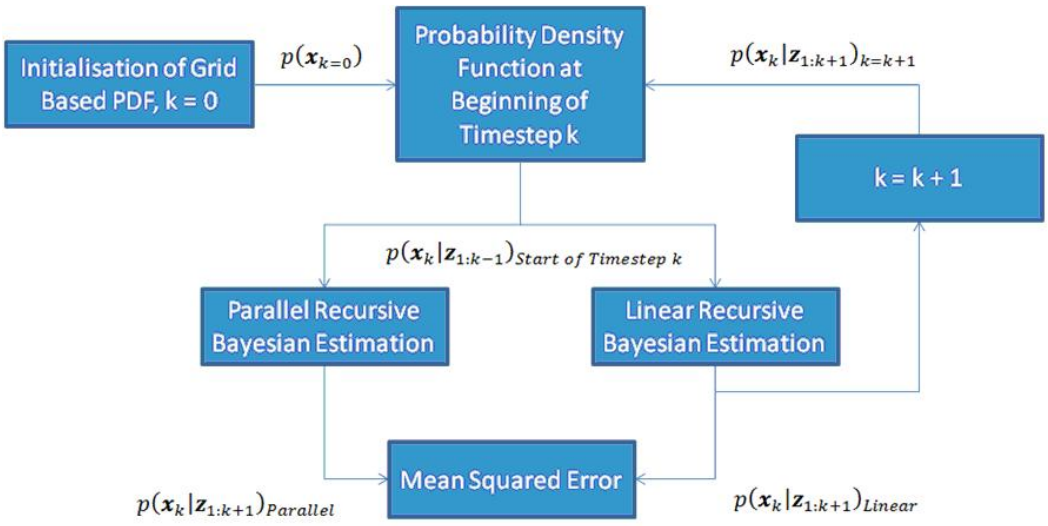


Fig. 5.1: Flow Chart for Comparison of Linear and Parallel Algorithms

The results for the mean squared error are shown in Figure 5.2.1. These indicate that there is a small random error between the two algorithms, with a mean value of 1.0 x 10-6. This is well within the inherent noise of the models being used in the iteration, and is of the correct order of magnitude to suggest that it is due to the use of single precision floating point operations on the GPU based convolution as opposed to the double precision floating point operations natively used by Matlab.

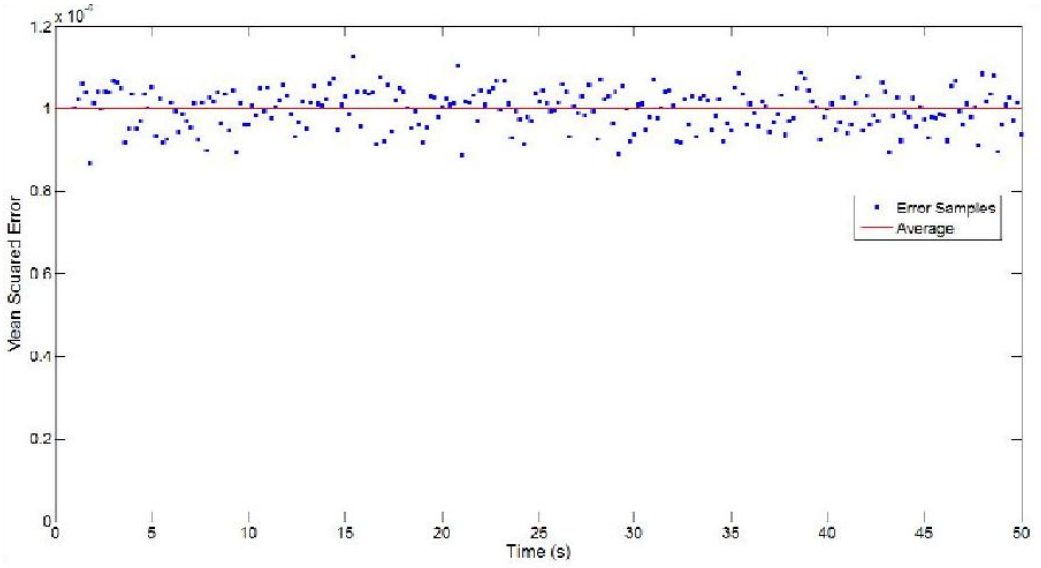


Fig. 5.2: Mean Squared Error for Parallel Recursive Bayesian Estimation

5.2.2 Test 2: Validation of Target Motion Model

The accuracy of the recursive Bayesian estimation was examined to determine the parameters required for its real-time computation. The size of this convolution kernel will affect both the accuracy of the representation of the targets motion as well as the real-time performance of the system itself. Realistic limits on the size of the convolution kernel need to be established to ensure that this accuracy is within the noise of the system to be implemented. In this test, the velocity characteristics of the motion model of the target were varied as well as the kernel size used to represent them.

Figure 5.3 shows the results of a quantitative analysis of the ability of the parallel form of the recursive Bayesian estimation in tests performed at a variety of different grid sizes and convolution kernel sizes with different input target motions models. In each test, the system was initialized and was allowed to iterate using the parallel recursive Bayesian estimation. The PDF output at the end of each time step was analyzed by calculating the weighted mean position of the peak.

The results well show that the relative error is very small (less than 1%) across grid sizes from 10,000 to 1,000,000 grid cells when a convolution kernel is used and appropriate velocities are used. However the error becomes very large if a smaller convolution is used and the targets velocity is quite large. Note that as the number of

grid cells increase to above 1,000,000 the convolution kernels become less accurate at representing the target motion model as the number of grid cells between the center of the kernel and the peak in the kernel becomes larger. If a kernel radius of 32 grid cells or greater is used and the grid size is between 10,000 and 1,000,000 cells the error for target motions up to 10 m/s is less than 2% which is well within the noise included in the target motion model.

5.3 Computational Speedup by Parallelization

In Chapter 4, the method of data parallelization was proposed in which the PDF at each cell in the grid-based representation can be computed in parallel, independently of the other grid cells and that as a result, a speedup in the computation time of the algorithm could be achieved. The required measurements of the iteration times for the linear and parallelized forms of the recursive Bayesian estimation were taken at different kernel radii which are shown in the first plot of Figure 5.3 whereas the computational speedup for the experimental iteration times is shown along with the estimation of the maximum speedup in the second plot.

The resulting improvement in the iteration rate for the recursive Bayesian estimation appears to be quite close to the theoretical limit to which data parallelization could achieve with the NVidia GPU used. On average across the range of convolution kernels tested, the NVidia GPU implementation achieved over 95% of the estimated maximum improvement in computational speed. This indicates that, as predicted, the high arithmetic intensity of the grid-based prediction step has resulted in the considerable improvement, and that the additional overhead due to memory access and idle threads created for coalesced data reads was small enough to not affect the result significantly.

The computational speedup should be independent of the number of grid cells in the search space as predicted in Chapter 4. To investigate the validity of this conclusion, the test was repeated at different grid sizes. The results shown in Figure 5.3 support the conclusion that the speedup achieved at different kernel radii is independent of the number of grid cells used. The figure also shows that the speedup

achieved for different kernel radii is above 95% of the estimated value.

5.4 Parametric Studies in Real-time Performance

The objective of this project is to develop a technique of performing recursive Bayesian estimation at an appropriate rate to be considered real-time so that it may be utilizable in real search and tracking operations and simulations involving other hardware and software systems. Parametric studies were used to specify the real-time performance of the recursive Bayesian estimation system so that users wishing to use the system can determine how the system will operate with their existing hardware and they can select the appropriate input parameters and scaling for their application. How the system performance scales with parameters such as the number of sensor platforms, search space size, prediction size and number of targets is very important for integration of this system with others.

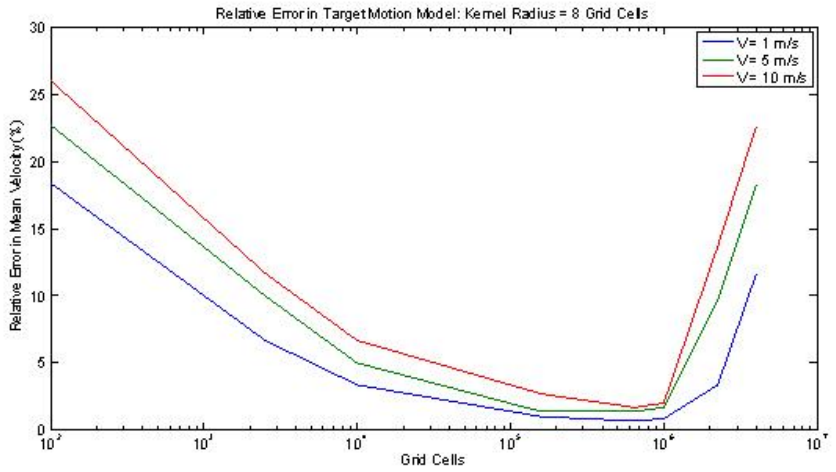
5.4.1 Search Space Grid Size

One of the most important parameter in determining the real-time performance of the system is the size of the grid space itself. As the number of grid cells increases, the number of floating point and memory operations, and in turn the iteration time, increases linearly. The iteration time for the real-time recursive Bayesian estimation system was measured at different sizes of grid spaces and the results are shown in Figure 5.4.1. The system shows reasonable real-time performance over this range of grid cells. Although the iteration time increases linearly with total number of grid cells, even at the largest grid size tested, 1,000,000, the iteration time is in the order of 0.1 seconds.

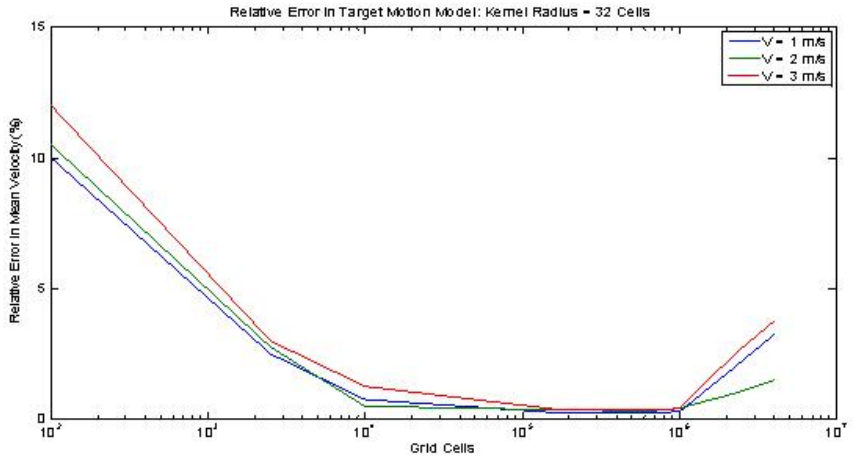
5.4.2 Number of Sensor Platforms and Sensor Range

Previous tests of the real-time performance were carried out with a single sensor platform with a typical search range, however the impact with which the characteristics and quantities searching sensor platforms have on the real-time performance

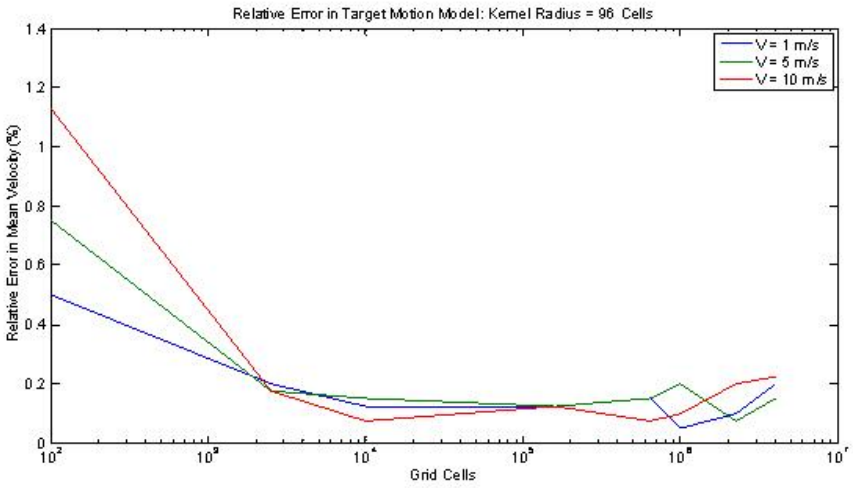
should also be analyzed. Figure 5.4.2 shows the resulting iteration time of the real-time recursive Bayesian estimation system for different numbers of sensor vehicles and with sensor ranges of 5m, 50m and 100m. The results illustrate that the update step in the recursive Bayesian estimation for small sensor ranges, the linear increase in time taken for the update step is minimal as the number of sensors being input increases. Even for sensor ranges as large as 100m, the iteration time for 100 such sensor platforms is only 20% longer than for a single sensor platform. These results indicate that the real-time Recursive Bayesian estimation system will still be able to iterate at real-time speeds for problems involving large numbers of search craft with considerable sensor ranges.



(a) 8 grid cells



(b) 32 grid cells



(c) 96 grid cells

Fig. 5.3: Parallel recursive Bayesian estimation

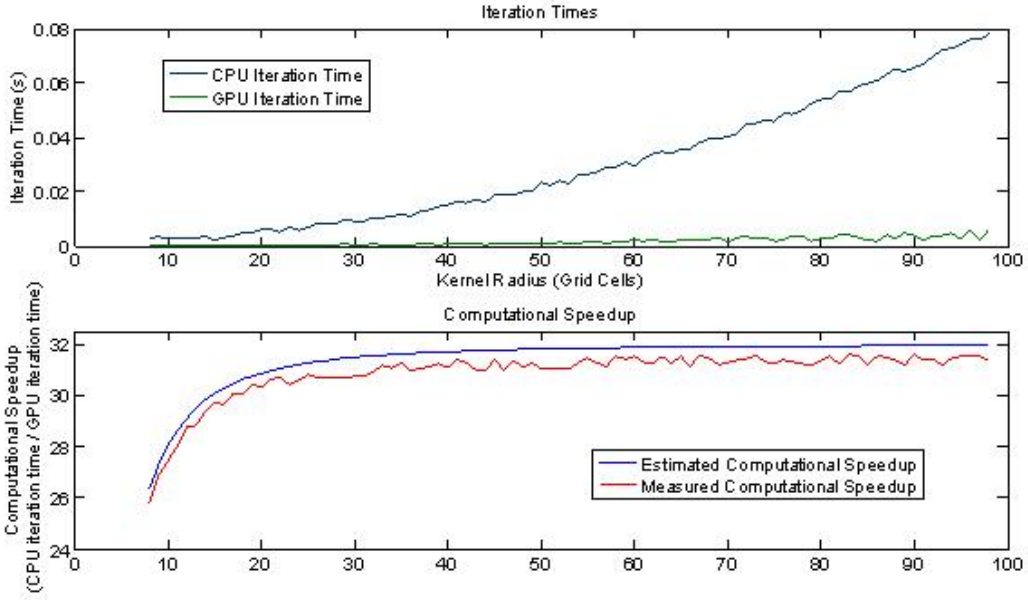


Fig. 5.4: Iteration times and computational speedup

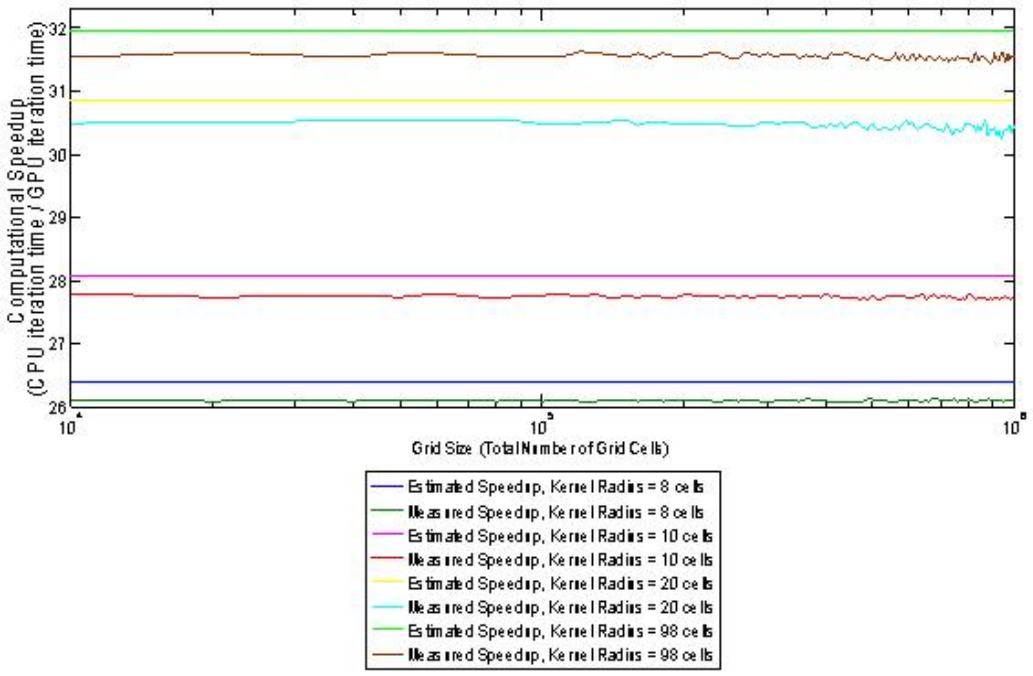


Fig. 5.5: Computational speedup vs size of grid space

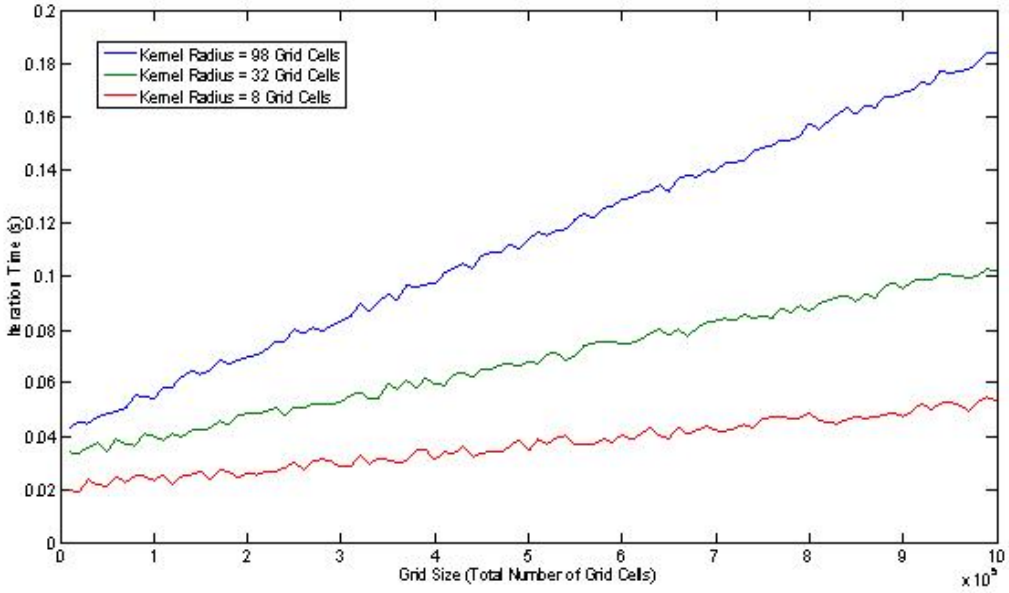


Fig. 5.6: Iteration time vs grid size

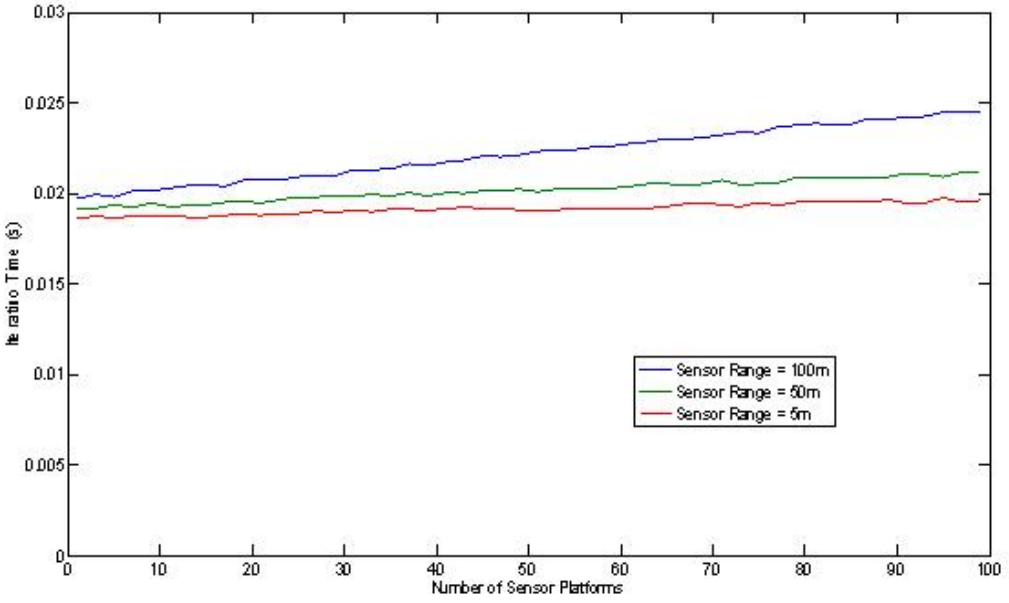


Fig. 5.7: Real-time performance with multiple sensor platforms